PCI Express® Architecture Transaction Layer Test Specification Revision 2.0

August 11, 2008

Revision	Revision History	DATE
2.0	Initial release.	08/11/2008

PCI-SIG® disclaims all warranties and liability for the use of this document and the information contained herein and assumes no responsibility for any errors that may appear in this document, nor does PCI-SIG make a commitment to update the information contained herein.

Contact the PCI-SIG office to obtain the latest revision of this specification.

Questions regarding the PCI Express Base Specification or membership in PCI-SIG may be forwarded to:

Membership Services
www.pcisig.com
E-mail: administration@pcisig.com
Phone: 503-619-0569
Fax: 503-644-6708

Technical Support
techsupp@pcisig.com

DISCLAIMER

This specification is provided "as is" with no warranties whatsoever, including any warranty of merchantability, noninfringement, fitness for any particular purpose, or any warranty otherwise arising out of any proposal, specification, or sample. PCI-SIG disclaims all liability for infringement of proprietary rights, relating to use of information in this specification. No license, express or implied, by estoppel or otherwise, to any intellectual property rights is granted herein.

Contents

1. INTRODUCTION 5

2. TEST ASSERTIONS 7

2.1. TABLE OF ASSERTIONS 7

3. GENERAL TEST OVERVIEW AND TOPOLOGY 11

4. MACROS 13

4.1. OVERVIEW 13

4.1.1. Integer MACRO_GetDUT_VCCapsStructure 13

4.1.2. Integer Macro Poll_HotPlugCmdComplete (Integer Slot_Num) 13

4.1.3. Integer MACRO Check_NativeHotPlug_Capable (Integer Slot_Num) 14

5. TEST DESCRIPTIONS 15

5.1.1. Test 1_2 TXN_BFT_RequestCompletion_UR 15

1

1. Introduction

This document contains a list of Test Assertions and a set of Test Definitions pertaining to the Transaction Layer. Assertions are statements of spec requirements which are measured by the algorithm details as specified in the Test Definitions. "Basic Functional Tests" are Test Algorithms which perform basic tests for key elements of Transaction Layer device functionality. This document does not describe a full set of PCI Express tests and assertions and is in no way intended to measure products for full design validation. Tests described here should be viewed as tools to checkpoint the result of product validation – not as a replacement for that effort.

Devices must also meet the applicable requirements and tests described in the following documents as well as any tests provided by the PCI-SIG:

PCI Express Architecture PHY Test Specification

PCI Express Architecture Configuration Space Test Specification

Platform BIOS Test Considerations for PCI Express Architecture

PCI Express Architecture Link Layer Test Specification

2

2. Test Assertions

Note: N/A in the "Test ID" column simply means that a Test Description which measures that Assertion is not included *with this revision* of this Test Document. A future revision of this (or another) Test Document *may* include Test Definitions to cover these Assertions. Regardless of whether or not a Test Description is provided, these Assertions still represent required criteria.

Note: There are published PCI Express 1.1 checklists. The 1.1 assertions listed here are the assertions from the checklists documents tested by configuration test descriptions. The checklists were not updated for PCI Express 2.0.

2.1. Table of Assertions

Assertion ID	Assertion Description	Test ID
Transaction Layer Protocol/Packet Definition		
TXN.2.0#2	All TLP fields marked Reserved "R" in the Base Specification must be filled with all 0's when a TLP is formed.	N/A
Transaction Descriptor – Transaction ID Field		
TXN.2.7#9	Functions must capture the Bus and Device Numbers supplied with all Configuration Write Requests (Type 0) completed by the function and supply these numbers in the Bus and Device Number fields of the Requester ID for all Requests initiated by the device/function.	Test 1.1
INTx Interrupt Signaling		
TXN.2.13#1	The INTx messages must adhere to the encoding defined in Table 2-12 of the Base Specification.	N/A
TXN.2.13#3	Assert_INTx/Deassert_INTx Messages do not include a data payload (TLP Type is Msg).	Test 2.1
TXN.2.13#4	The TLP Header Length field is reserved for INTx Signaling messages.	Test 2.1
TXN.2.13#7	Assert_INTx and Deassert INTx interrupt Messages must use the default Traffic Class designator (TC0).	Test 2.1
TXN.2.13#10	An Assert_INTx represents the active going transition of the INTx (x = A, B, C, or D) virtual wire.	Test 2.1
TXN.2.13#11	A Deassert_INTx represents the inactive going transition of the INTx (x = A, B, C, or D) virtual wire.	Test 2.1

Assertion ID	Assertion Description	Test ID
TXN.2.13#13	INTx Interrupt Signaling is disabled *with* the Interrupt Disable bit of the Command register.	Test 2.1
TXN.2.13#14	Any INTx virtual wires that are active when the Interrupt Disable bit is set must be deasserted by transmitting the appropriate Deassert_INTx Message(s).	Test 2.1
TXN.2.13#21	The Requester ID of an Assert INTx/Deassert INTx Message will correspond to the Transmitter of the message on that Link (not necessarily to the original source of the interrupt).	Test 2.1
Error Signaling Messages		
TXN.2.15#1	Error Signaling Messages must comply with the encoding in Base Specification Table 2-15.	N/A
TXN.2.15#2	Error Signaling Messages do not include a data payload (TLP Type is Msg).	Test 2.4
TXN.2.15#3	Within Error Signaling Messages the TLP Header Length field is Reserved.	Test 2.4
TXN.2.15#4	Error Signaling Messages must use the default Traffic Class designator (TC0).	Test 2.4
Native Hot Plug Signaling		
TXN.2.19#4	Hot Plug Signaling Messages must comply with the encoding in Base Specification, Table 2-19.	N/A
TXN.2.19#1	Components must accept received hot plug signaling messages as defined in the Support column of Base Specification, Table 2-19, without indicating an error.	Test 2.3
Completions		
TXN.2.21#15	Functions must capture the Bus and Device Numbers supplied with all Type 0 Configuration Write Requests completed by the function, and supply these numbers in the Bus and Device Number fields of the Completer ID for all Completions generated by the device/function.	Test 1.1
TXN.2.21#19	Completion headers must supply the same values for the Requester ID, Tag, Attribute and Traffic Class as were supplied in the header of the corresponding Request	Test 1.1

Assertion ID	Assertion Description	Test ID
Request Handling		
TXN.3.1#1	A Request Type which is not supported by the device is an Unsupported Request and is reported according to Base Specification Section 6.2.	Test 1.1
TXN.3.1#2	For Unsupported Requests requiring Completion, a Completion Status of UR is returned.	Test 1.1
Data Return for Read Requests		
TXN.3.2#32	When a Read Completion is generated with a Completion Status other than Successful Completion no data is included with the Completion and the Cpl (or CplLk) encoding is used instead of CplD (or CplDLk).	Test 1.1
Virtual Channel (VC) Mechanism		
TXN.5.0#3	PCI Express Completers that do not implement the optional PCI Express Virtual Channel Capability Structure must accept requests with TC label other than TC0.	Test 5.1
TXN.5.0#4	PCI Express Completers that do not implement the optional PCI Express Virtual Channel Capability Structure must generate Completions with the same TC label as the label of the Request.	Test 5.1
TXN.5.0#5	Switches that do not implement the optional PCI Express Virtual Channel Capability Structure must map all TCs to VC0.	N/A
TXN.5.0#6	Switches that do not implement the optional PCI Express Virtual Channel Capability Structure must forward all transactions regardless of the TC Label.	N/A
Virtual Channel Identification (VC ID)		
TXN.5.1#1	All PCI Express Ports that support more than VC0 must provide the VC Capability Structure according to the definition in Base Specification Section 7.11.	Test 3.1
TXN.5.1#4	VC ID 0 is assigned and fixed to the default VC.	N/A
Flow Control Rules		
TXN.6.1#1	Flow Control information is transferred using Flow Control Packets (FCPs)(see Base Specification Section 3.4).	Test 3.1
TXN.6.1#14	InitFC1 and InitFC2 FCPs are used for Flow Control initialization (see Base Specification, Section 3.3).	Test 3.1
TXN.6.1#16	During FC initialization for any Virtual Channel, including the default VC initialized as a part of Link initialization, Receivers must initially advertise VC credit values equal to or greater than those shown in Base Specification, Table 2-27.	Test 3.1

Assertion ID	Assertion Description	Test ID
Completion Timeout Mechanism		
TXN.8.0#1.1	Completion Timeout support is required of Endpoint devices	Test 1.2
TXN.8.0#2	The PCI Express elements that are capable of initiating Requests that invoke Completions must implement Completion Timeout mechanism. An exception is made for Configuration Requests.	Test 1.2
TXN.8.0#3	The Completion Timeout timer must not expire (i.e., cause a timeout event) in less than 50 µs. It is strongly recommend that unless an application requires this level of timer granularity the minimum time should not expire in less than 10 ms.	Test 1.2
TXN.8.0#4	The Completion Timeout timer must expire if a Request is not completed in 50 ms.	Test 1.2
TXN.8.0#5	A Completion Timeout is a reported error associated with the Requestor device/function.	Test 1.2
Link Width Negotiation		
EM.6#4	A x8 add-in card (and endpoint) must operate as a x4 card (and endpoint) when plugged into a x8 connector that has only the first four lanes routed.	Test 3.4
EM.6#3	All PCI Express cards and slots must meet the interoperability requirements as defined in Section 6.3 of the PCI Express Card Electromechanical Specification.	Test 3.4

3

3. General Test Overview and Topology

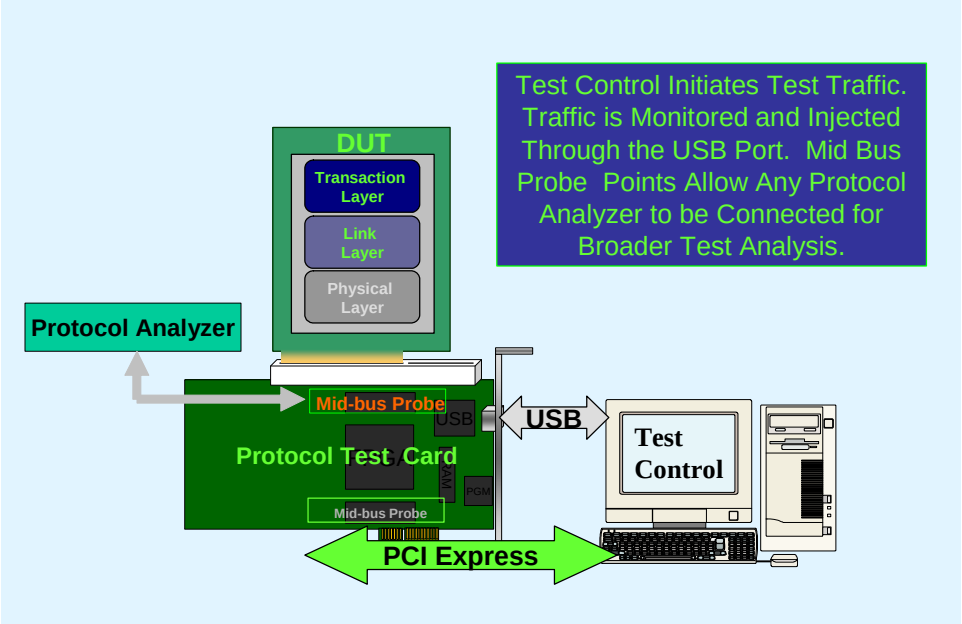

Figure 1: Transaction Layer Basic Test Topology

4

4. MACROS

4.1. Overview

4.1.1. Integer MACRO_GetDUT_VCCapsStructure

Description: Read the VC Capabilities Structure from the DUT and return a count of VCs supported by the DUT beyond the required/default VC0.

Algorithm :

- Attempt to locate the *Virtual Channel Capability Structure* in DUT configuration space
- If found {
  - Read the *Extended VC Count* from the *Port VC Capability register 1*
  - Return (VC Count +1)
- } Else Return (0)

4.1.2. Integer Macro Poll_HotPlugCmdComplete (Integer Slot_Num)

Description: This macro polls the *Slot Status register* of the Slot number *Slot_Num* for Native Hot Plug command complete status (bit offset 4).

Input Arguments: Integer *Slot_Num* – the physical number of the slot to check

Return Values: Integer

1 = slot at *Slot_Num* has posted command complete

0 = slot at *Slot_Num* has not posted command complete

-1 = *Slot_Num* exceeds highest numbered slot within chassis

Algorithm:

- Scan the PCI Express Capabilities lists to locate each of the slots in the system {
  - Read the *Slot Capabilities register* to determine if it's a match for *Slot_Num* {
    - If it is a match for *Slot_Num* {
      - Poll the *Slot Status register* for command complete (bit 4) for 1 second

- If command complete not set within 1 second
  - return (0)
- Else {
  - Write 0 to the *Slot Status register* bit offset 4 to clear command complete
  - return (1)
  - }

} else continue slot scan

} continue scan

} Scan complete, unable to locate *Slot_Num* – return (-1)

4.1.3. Integer MACRO Check_NativeHotPlug_Capable (Integer Slot_Num)

Description: This macro expects that physical slots are sequentially numbered from 1 to the highest numbered slot within the chassis (physical slots are prevented by the Base Specification from being assigned a value of 0). The macro checks the physical slot number *Slot_Num* to see if it supports Native Hot Plug.

Input Arguments: Integer *Slot_Num* – the physical number of the slot to check

Return Values: Integer

1 = slot at *Slot_Num* supports Native Hot Plug

0 = slot at *Slot_Num* does not support Native Hot Plug

-1 = *Slot_Num* exceeds highest numbered slot within chassis

Algorithm:

- Initialization: scan the PCI Express Capabilities lists to locate all of the slots in the system
- Initialization: read the *Slot Capabilities registers* of all slots and build a static *NativeHotSlot_Num* table of all Native Hot Plug capable slots in the system – put the slot numbers of those slots in this table
- When called: search the *NativeHotSlot_Num* table for *Slot_Num* and return the appropriate value from above Return Values list

5

5. Test Descriptions

All tests are run at both 2.5 GT/s and 5.0 GT/s for add in cards that support 5.0 GT/s.

5.1.1. Test 1_2 TXN_BFT_RequestCompletion_UR

The intent of this test is to verify that the DUT will issue a UR completion for the configuration requests that it does not support. In addition, depending on the role-based error reporting will send the appropriate error messages to the root complex. At this point the algorithm applies to single function devices only. It may be expanded to handle MFDs in future when they are expected to be available.

ASSERTIONS COVERED: TXN.02.07#029.00, TXN.02.21#15.00

NOTES:
Test applies to all PCI Express Single Function Devices.

DATA_BUF – holds the data read back from the device.

5.1.1.1. RC Test

TOPOLOGY: Platform Test Topology, PTC in Platform Test mode

SECTION NOTES:

1) RC (DUT) is the CONFIG_RD requester and PTC is the completer for that request in this test.

INITIAL CONDITIONS:

1) Platform is up and running, with drivers for PTC loaded and functioning.
2) PTC is disarmed and no trigger conditions set up.

PROCEDURE:

TO BE DONE LATER

5.1.1.2. Endpoint Device Test

TOPOLOGY: Endpoint Test Topology, PTC in Add-in card Test mode

SECTION NOTES:

1) RC is the CONFIG_RD requester and EP (DUT) is the completer for that request in this test.

INITIAL CONDITIONS:

1) Platform is up and running, with drivers for the PTC loaded and functioning.
2) PTC is disarmed and no trigger conditions set up.
3) DUT is running default traffic (if any) – that is, no application started yet.

PROCEDURE:

1) Clear Device Status register by writing to the Device Control register of the DUT and verify that none of the error status bits are set
2) MACRO_PTC_CONFIG_TRACE_BUF(TLP_HEADERS ONLY, UPSTREAM_DIR)
3) MACRO_PTC_PROGRAM(DELAY_ACK_NAK_LEGAL, CFG_RD, 1) // just to kick start the trace buffer in the PTC and is a benign command to act on
4) MACRO_PTC_ARM () // starts the trace buffer capture of all TLP headers from DUT
5) For (Function=0 to Function7)
   a. DATA_BUF = MACRO_READ_CONFIG_DATA_FROM_DUT (VENDOR_DEVICE_ID) // expectation is, not all eight functions are implemented in any DUT. In the case they are, the test will abort indicating as such and is not considered a valid test for that device.
6) MACRO-PTC_DISARM ()
7) MACRO_READ_DATA_FROM_PTC( )

CASE 1: PCIe1.0.a - Non-Fatal Error - Severity – Non-Fatal - AER implemented

This implies the DUT implemented AER and, in the Uncorrectable Severity register, UR bit (bit 20) is cleared to indicate that this is *not* a fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Non-Fatal Error bit (bit 1) in the DUT's Device Status register is set.
   c. Unsupported Request bit (bit 3)) in the DUT's Device Status register is set.

d. If the error is *not* masked in the Uncorrectable Error Mask register (bit 20) of AER
   i. UR bit (bit 20) in the Uncorrectable Error Status register of the AER is set.
   ii. If UR error reporting is enabled in DUT's Device Control register, send ERR_NONFATAL message.
   iii. If UR error reporting is *not* enabled in DUT's Device Control register, *no* ERR_NONFATAL is sent.
e. If the error is masked in the Uncorrectable Error Mask register (bit 20) of AER then *no* ERR_NONFATAL is sent

2) If all of the conditions in the above step are met, DUT PASSES the test. Otherwise consider it as DUT's failure.

CASE 2: PCIe1.0.a - Non-Fatal Error - Severity – Fatal - AER implemented

This implies DUT implemented AER is and in the Uncorrectable Severity register, UR bit (bit 20) is set to indicate that this is a fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Fatal Error bit (bit 2) in the DUT's Device Status register is set.
   c. Unsupported Request bit (bit 3)) in the DUT's Device Status register is set.
   d. UR bit (bit 20) in the Uncorrectable Error Status register of the AER is set.
   e. If the error is *not* masked in the Uncorrectable Error Mask register (bit 20) of AER.
      i. If UR error reporting is enabled in DUT's Device Control register, ERR_FATAL message is sent.
      ii. If UR error reporting is NOT enabled in DUT's Device Control register, *no* ERR_MSG is sent.
   f. If the error is masked in the Uncorrectable Error Mask register (bit 20) of AER, then *no* ERR_FATAL is sent.
2) If all of the conditions in the above step are met DUT PASSES the test. Otherwise consider it as DUT's failure.

CASE 3: PCIe1.0.a - Non-Fatal Error - No AER

This implies DUT has no AER and the UR is handled as a non-fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Non-Fatal Error bit (bit 1) in the DUT's Device Status register is set.
   c. Unsupported Request bit (bit 3)) in the DUT's Device Status register is set.
   d. If UR error reporting is enabled in DUT's Device Control register, send ERR_NONFATAL message.
      i. If UR error reporting is *not* enabled in DUT's Device Control register, *no* ERR_NONFATAL is sent.
2) If all of the conditions in the above step are met, DUT PASSES the test. Otherwise consider it as DUT's failure.

CASE 4: PCIe1.1 - Non-Fatal Error - Severity – Advisory - AER implemented

This implies the DUT implemented AER and, in the Uncorrectable Severity register, UR bit (bit 20) is cleared to indicate that this is *not* a fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Correctable Error bit (bit 0) in the DUT's Device Status register is set and Non-Fatal Error bit (bit 1) and Fatal Error bit (bit 2) are clear.
   c. Unsupported Request bit (bit 3)) in the DUT's Device Status register is set.
   d. Advisory Non-Fatal bit (bit 13) in the Correctable Error register of the AER is set.
   e. If the error is *not* masked in the Correctable Error Mask register (bit 13) of AER
      i. UR bit (bit 20) in the Uncorrectable Error Status register of the AER is set.
      ii. If UR error reporting is enabled in DUT's Device Control register, send ERR_COR message.
      iii. If UR error reporting is *not* enabled in DUT's Device Control register, *no* ERR_COR is sent.
   f. If the error is masked in the Correctable Error Mask register (bit 13) of AER, then *no* ERR_COR is sent.
2) If all of the conditions in the above step are met, DUT PASSES the test. Otherwise consider it as DUT's failure.

CASE 5: PCIe1.1 - Non-Fatal Error – NO AER implemented

This implies DUT has no AER and the UR is handled as a non-fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Non-Fatal Error bit (bit 1) in the DUT's Device Status register is set.
   c. Unsupported Request bit (bit 3)) in the DUT's Device Status register is set.
   d. No ERR_NONFATAL message is sent.
2) If all of the conditions in the above bullet are met, DUT PASSES the test. Otherwise consider it as DUT's failure.

CASE 6: PCIe1.1 – Fatal Error – Severity – Non-Advisory - AER implemented

This implies the DUT implemented AER and, in the Uncorrectable Severity register, UR bit (bit 20) is set to indicate that this is a fatal error.

1) Verify that:
   a. The DUT sent one or more UR completions for all functions not implemented.
   b. Fatal error bit (bit 2) in the DUT's Device Status register is set and Non-Fatal Error bit (bit 1) and Correctable Error bit (bit 0) are clear.
   c. UR bit (bit 3)) in the DUT's Device Status register is set.
   d. UR bit (bit 20) in the DUT's Uncorrectable Error Status register is set.
   e. If the error is *not* masked in the Uncorrectable Error Mask register (bit 20) of AER
      i. If UR error reporting is enabled in DUT's Device Control register or SERR in Command register is set, send ERR_FATAL message.
      ii. If UR error reporting is *not* enabled in DUT's Device Control register or SERR in Command register is not set, *no* ERR_FATAL message is sent.
   f. If the error is masked in the Uncorrectable Error Mask register (bit 20) of AER, then *no* ERR_FATAL message is sent
2) If all of the conditions in the above step are met, DUT PASSES the test. Otherwise consider it as DUT's failure.

5.1.1.3. Switch Downstream Port Test

TOPOLOGY: Switch Test Topology, PTC in Platform Test mode

SECTION NOTES:

1) Algorithm same as in the RC test case except the DUT is a Switch's downstream port.

5.1.1.4. Switch Upstream Port Test

TOPOLOGY: Endpoint Test Topology, PTC in Add-in card Test mode

SECTION NOTES:

1) Algorithm same as in the Endpoint device test case except the DUT is the Switch's upstream port.